Softwareentwicklung II (IB)

Blatt 4

Prof. Dr. Oliver Braun

Fakultät für Informatik und Mathematik
Hochschule München

Letzte Änderung: 19.05.2018 23:27

Abgabe der Aufgabe auf diesem Blatt: bis 13.06.18, 08:00 Uhr durch Pushen in das zur Aufgabe gehörende GitHub-Repository.

Aufgabe 1 — Filmverleih

Das Repository für diese Aufgabe bekommen Sie unter https://classroom.github.com/a/G3BXhN2e.

a) **Fsk**

Implementieren Sie die Enum `Fsk`, die folgende Eigenschaften hat:

- Es gibt die fünf Enumwerte: `FSK0`, `FSK6`, `FSK12`, `FSK16` und `FSK18`.
- Es gibt eine private Objektvariable `age`, die das Alter speichert **ab** dem die jeweilige FSK-Altersfreigabe zugelassen ist. Hinweis: Sie brauchen dann auch einen Konstruktor.
- Methoden:
  - Überprüfen ob das Alter für die Altersfreigabe zulässig ist:

    ```
    public boolean ageOk(final int age)
    ```

  - Berechnen der höchsten, für das übergebene Alter zulässigen Altersfreigabe:

    ```
    public static Fsk getFskForAge(final int age)
    ```

Diese Funktion soll auch bei (unsinnigen) negativen Werten die `FSK0` zurück geben.

- Redefinieren Sie die `toString`-Methode

  ```
  @Override public String toString()
  ```

  Auch wenn die Enum-Klasse schon eine `toString`-Methode generiert, wollen wir eine eigene bei der zwischen `FSK` und dem Alterswert ein Leerzeichen ist, also `FSK 0` statt `FSK0`.

b) **Movies**

Implementieren Sie die **unveränderliche** Klasse `Movie`, die folgendes beinhaltet:

- private Objektvariablen mit öffentlichen Gettern

  `title` der Titel des Films
  `fsk` die FSK-Freigabe des Films als `Fsk`-Wert

- Konstruktor:

  Ein Konstruktor der den Title und das Freigabe-Alter als `int` als Parameter hat.

- Methode:
  - Redefinieren Sie die Methode `toString` so, dass ein Film wie folgt als `String` repräsentiert wird:

    Der Titel gefolgt von einem Leerzeichen und der FSK-Freigabe in runden Klammern.

  - Redefinieren Sie die Methoden `equals` und `hashCode`.

c) **Customer**

Implementieren Sie die Klasse `Customer` mit folgenden Eigenschaften:

- private Objektvariablen (zunächst ohne Getter!)

  `name` der Name eines Kunden

  `birthday` der Geburtstag eines Kunden. Nutzen Sie dazu den Typ `java.time.LocalDate`.

  `rentedMovies` eine Map der ausgeliehenen Filme:

  ```
  private final Map<Integer, Movie> rentedMovies = new HashMap<>();
  ```

- einen Konstruktor

  ```
  public Customer(
      final String name,
      final int dayOfBirth,
      final int monthOfBirth,
      final int yearOfBirth)
  ```

Die Parameter entsprechen dem Namen, sowie dem Geburtstag getrennt in Tag, Monat und Jahr, als `int`, also z.B.

```
new Customer("James Gosling", 19, 5, 1955);
```

Um aus den drei `int`s einen Wert vom Typ `LocalDate` zu machen, nutzen Sie im Konstruktor einfach die Methode

```
LocalDate.of(yearOfBirth, monthOfBirth, dayOfBirth);
```

die ein `LocalDate`-Objekt zurück gibt, das das Datum repräsentiert. Achtung: Die Reihenfolge Tag, Monat, Jahr ist bei `of` genau umgedreht.

- folgende öffentliche Getter

  `getName` gibt den Namen zurück
  `getAge` gibt das Alter in Jahren zurück. Statt das Alter wie im ersten Semester selbst zu berechnen, können Sie die statische Methode `between` aus der Klasse `java.time.Period` benutzen. Erster Parameter ist das Geburtstagsobjekt, zweiter das heutige Datum (`LocalDate.now()`), Ergebnis ist ein `Period`-Objekt das die Anzahl von Tagen, Monaten und Jahren seit dem Geburtstag enthält. Die Anzahl der Jahre bekommen Sie dann von dem `Period`-Objekt mit dem Getter `getYears`.

- die Methode

  ```
  public int getRentedMoviesCount()
  ```

  die die Anzahl der vom Kunden zur Zeit ausgeliehenen Filme zurück gibt. Tipp: Fragen Sie einfach die Map nach der Anzahl ihrer Elemente.

- die Methode

  ```
  public List<String> getRentedMoviesList()
  ```

  die eine Liste von Titeln der vom Kunden zur Zeit ausgeliehenen Filme zurück gibt. Nutzen Sie z.B. eine `ArrayList`.

- zum Ausleihen eines Films dient die Methode

  ```
  public boolean rentMovie(final int serial, final Movie movie)
  ```

  Nur wenn der Film für das Alter des Kunden zugelassen ist, soll er in die `rentedMovies`-Map eingefügt werden. Dann ist das Ergebnis `true`, sonst `false`.

- zum Zurückgeben eines Filmes gibt es die Methoden

  ```
  public Movie returnMovie(final int serial)
  public int returnMovie(final String title)
  ```

  Bei der einen wird ein Film über seine Seriennummer, bei der anderen über seinen Titel referenziert. In beiden Fällen soll der jeweilige Film, wenn er tatsächlich ausgeliehen ist, aus der `rentedMovies`-Map entfernt werden. Die

eine Methode gibt dann den Film, die andere die Seriennummer des Films zurück. Sollte er nicht in der Map enthalten sein, wird `null` bzw. `-1` zurück gegeben.

- die Methode `toString` soll den Namen, gefolgt von einem Leerzeichen und dem Alter in Klammern zurück geben. Vor dem Alter in Jahren soll stehen `Alter:` und ein Leerzeichen, also z.B. `Baby (Alter: 0)`.

d) **Result**

Implementieren Sie eine Enum-Klasse `Result` mit den Enumwerten `UserDoesNotExist`, `UserTooYoung`, `UserMaximumMoviesReached`, `MovieNotAvailable`, `MovieNotRented`, `MovieRentedButUserNotFound`, `Success`. Diese Werte dienen als Ergebnisse in der folgenden Klasse.

e) **MovieStore**

Implementieren Sie die Klasse `MovieStore` mit folgenden Eigenschaften:

- private Klassenvariablen
  - `lastMovieSerial` und `lastUserId`, die beide bei 0 starten

  mit statischen Methoden
  - `getNextMovieSerial` und `getNextUserId`, die den jeweiligen Zähler inkrementieren und den nächsten Wert zurück geben.
- private Objektvariablen

  ```
  private Map<Integer, Movie> availableMovies = new HashMap<>();
  private Map<Integer, Movie> rentedMovies = new HashMap<>();
  private Map<Integer, Customer> customers = new HashMap<>();
  ```

  für die Verwaltung der verfügbaren und ausgeliehenen Filme sowie der Kunden. Der Schlüssel ist bei den Filmen die Seriennummer und bei den Kunden die UserId.

  Außerdem gibt es eine private Objektvariable `maxRentableMoviesByCustomer` die die Anzahl der Filme angibt, die ein Kunde maximal auf einmal aus geliehen haben darf. Für diese Objektvariable gibt es einen Getter und einen Setter. Der Setter läßt den Wert unverändert, wenn der Parameter kleiner oder gleich 0 ist.
- einen Konstruktor, der `maxRentableMoviesByCustomer` als Parameter hat und setzt. Ist der übergebene Wert kleiner oder gleich 0, wird er auf 1 gesetzt.
- eine **private** Methode

  ```
  private int getAvailableMovieId(final String title)
  ```

  die die Seriennummer eines Filmes zurück gibt der den übergebenen Title trägt und ausleihbar ist (d.h. in `availableMovies`). Wird kein Film mit dem

Namen gefunden, soll `-1` zurück gegeben werden, also eine Seriennummer, die es nicht geben kann.

- eine öffentliche Methode

  ```
  public Set<Movie> getAvailableMovies()
  ```

  Die eine Menge der ausleihbaren Filme zurück gibt. Obwohl es einen Film mehrfach (mit verschiedenen Seriennummern) geben kann, wird er in einem `HashSet` nur einmal vorhanden sein (sofern Sie `equals` und `hashCode` in der Klasse `Movie` korrekt implementiert haben). Nutzen Sie also zum "aufsammeln" ein `HashSet`, z.B.

  ```
  Set<Movie> movies = new HashSet<>();
  ```

  und fügen Sie einfach alle verfügbaren Filme (aus `availableMovies`) in `movies` ein.

- einen öffentliche Methode

  ```
  public Customer getCustomer(final int userId)
  ```

- eine öffentliche Methode

  ```
  public int addMovie(final String title, final int fskAge)
  ```

  die einen Film mit dem übergebenen Titel und der Altersfreigabe zu den verfügbaren Filmen mit der nächsten Seriennummer hinzufügt und diese als Ergebnis zurück gibt.

- eine öffentliche Methode

  ```
  public List<Integer> addMovies(final String title,
                                 final int fskAge,
                                 final int count)
  ```

  Diese fügt einen Film `count`-mal zu den verfügbaren Filmen hinzu. Nachdem der `Movie` unveränderbar ist, können Sie `count`-mal das selbe Objekt mit einer jeweils anderen Seriennummer verwenden.

  Die Seriennummern sammeln Sie in einer Liste:

  ```
  List<Integer> serials = new ArrayList<>();
  ```

  und geben diese als Ergebnis zurück.

- eine öffentliche Methode

  ```
  public int addCustomer(final String name,
                         final int dayOfBirth,
                         final int monthOfBirth,
                         final int yearOfBirth)
  ```

  die einen `Customer` erzeugt, zu den `customers` mit der nächsten freien UserId hinzufügt und die UserId dann als Ergebnis zurück gibt.

- eine öffentliche Methode

  ```
  public Result rentMovie(final int userId, final String movieTitle)
  ```

  die folgendermaßen vorgeht:

  - Gibt es keinen Kunden mit der UserId, ist das Ergebnis `UserDoesNotExist`.
  - Hat der Kunde schon so viele Filme ausgeliehen, wie maximal erlaubt, ist das Ergebnis `UserMaximumMoviesReached`.
  - Gibt es keinen ausleihbaren Film mit dem Titel, ist das Ergebnis `MovieNotAvailable`. Dabei ist es uninteressant, ob es vielleicht doch so einen Film gibt, aber der gerade ausgeliehen ist.
  - Erfüllt der Kunde die Altersfreigabe nicht, ist das Ergebnis `UserTooYoung`. Hinweis: Das wird ja schon von `rentMovie` in der Klasse `Customer` überprüft. Sie können also einfach versuchen den Film auszuleihen.
  - Passt alles, wird der Film ausgeliehen. D.h. er muss beim Kunden entsprechend gespeichert sein und muss von `availableMovies` nach `rentedMovies` verschoben werden. Als Ergebnis wird dann `Success` zurück gegeben.

- eine öffentliche Methode

  ```
  public Result returnMovie(final int movieSerial)
  ```

  die folgendermaßen vorgeht:

  - Wird der Film in `rentedMovies` nicht gefunden, ist das Ergebnis `MovieNotRented`.
  - Sonst wird bei allen Kunden probiert den Film auszutragen. Hinweis: Die Methode `returnMovie` aus der Klasse `Customer` kann dafür einfach ausgerufen werden. War es erfolgreich (was nur bei einem Kunden so sein sollte), muss der Film nur noch von `rentedMovies` wieder nach `availableMovies` verschoben werden. Ergebnis ist dann `Success`.
  - War die Rückgabe bei keinem Kunden erfolgreich, liegt ein Problem im System vor, dass eigentliche gar nicht auftreten kann. Trotzdem verschieben wir den Film dann von `rentedMovies` nach `availableMovies` und geben aber als Ergebnis `MovieRentedButUserNotFound`. Dieser Fall kann übrigens gar nicht getestet werden, da eine solche Situation mit den vorhandenen Methoden gar nicht erzeugt werden kann. Dennoch ist es sinnvoll, denn vielleicht passiert so etwas mal in einer zukünftigen Version der Software.

- eine öffentliche Methode

  ```
  public Result returnMovie(final int userId, final String movieTitle)
  ```

  die folgendermaßen vorgeht:

  - Wird kein Kunde mit der UserId gefunden, ist das Ergebnis `UserDoesNotExist`.

- Hat dieser Kunde keinen Film mit diesem Titel ausgeliehen, ist das Ergebnis `MovieNotRented`.
- Ansonsten Film verschieben und `Success` zurück geben.